COMPUTER ART : UNE PARTITION POUR LE PEINTRE

P.L. NEUMANN

INTRODUCTION

De l'idée à "l'expression" d'une oeuvre par ordinateur il y a un programme. Si ce programme n'est qu'un exécutant la création et donc "l'expression" de l'artiste ne s'exerceront qu'au niveau du programme. Pour qu'il y ait "expression" dans le résultat il faudrait que le programme soit l'artiste ou que l'artiste travaille avec son programme.

C'est dans cette dernière perspective que se situe le programme "INTERPRETATION".

Ce programme ne réalise rien, mais propose un programme pour l'artiste : une partition à interpréter.

Ce programme a été écrit en CONNIVER - LISP en Mars 1974.

Comment fonctionne ce programme ?
On lui donne une idée de base ou contexte de la forme :

(A sur B)
(C devant A)

c'est à dire un couple d'objets non définis, liés par une relation x quelconque telle qu'il existe son contraire $\bar{x}$.
A partir de cette ébauche le programme va développer une partition c'est à dire un certain nombre de propositions telles que (A x B) - A, B et x éléments du contexte.
Chaque étape de cette partition correspondra donc au développement d'un objet du contexte par rapport à un autre objet de ce même contexte, le choix d'un tel ou tel objet et d'une relation ne dépendant que de l'état courant du contexte à un instant donné.
En effet, chaque proposition du programme devra être cohérente par rapport à l'idée de base : l'union de la proposition (p) et du contexte (c) devra être égale au contexte (c) lui - même, puis (p) sera ajouté à (c), produisant ainsi un nouveau contexte permettant le développement de nouvelles relations.
Le processus se déroulera ainsi jusqu'à ce que l'idée de base n'offre plus aucune possibilité de développement.

Le second travail va maintenant consister à interpréter la partition ainsi construite : interpréter voulant dire dans le cas présent matérialiser les objets et les relations de la partition en vue d'une exécution en pas à pas, respectant l'ordre chronologique de cette partition.
Autrement dit, il s'agit en fait de concrétiser les éléments d'un algoritme produit par un programme et dont on aura auparavant précisé l'idée de base, puis l'exécuter à la main ou par programme.

LE PROGRAMME

I - Développement de l'idée de base ou contexte.

a) L'idée de base

Prenons comme exemple la liste des propositions suivantes correspondant pour le peintre à une esquisse de son dessin :

(A à gauche de B)
(A à gauche de C)
(B sur C)

et la liste de toutes les relations employées et de leurs inverses:

```
( Setq SIT ' ( à gauche de à droite de sur sous ))
```

.../...

Le programme va commencer par analyser ce contexte en y ajoutant les informations suivantes :

1 - Hypothèse de Travail:
" ne sont définis par relation que les objets A et B , en effet dans une relation du type (α x β) j'ai supposé que l'objet B était l'objet " subissant " la relation x et que donc, seul l'objet α était défini."

2 - Etat des Objets mis en relation:
(/ à droite de A) il doit y avoir quelquechose
(/ sous B) " " " " "
C n'est pas défini.

3 - Idée ou Pattern à respecter:
le graphe

A ⟶ B, A ⟶ C, B ⟶ C

ou la liste

((A B) (A C) (B C))

4 - Une liste comportant tous les objets et leur relation qui permettra de déterminer les objets à développer. Cette liste ne fera pas partie du contexte : elle représentera l'état de chaque objet.

(A est à gauche) de quelquechose
(B est à droite) " "
(B est sur) " "
(C est sur) " "
(C est à droite) " "

Après analyse le contexte initial sera représenté par la liste:

((A à gauche de B)
(A à gauche de C)
(B sur C)
(/ à droite de A) (/ sous B)
(A B) (A C) (B C))

b) Développement du contexte

Cette tâche va consister à produire une ou plusieurs propositions du type (α x β) respectant le pattern du contexte. Pour cela, le programme va se poser un certain nombre de questions.

1 - Quel objet développer et vers quel autre ?

Pour cela on ira consulter la liste de l'etat des objets (mentionnée en 4- du a)). Les objets a développer seront les objets qui auront le même état. En suivant notre exemple, ces objets seront B et C puisqu'ils possédent le même état: est à droite de .

.../...

Par ailleurs on s'assurera que la proposition (C B) n'appartient pas déjà au pattern du contexte. Dans le cas contraire on inverse cette proposition qui deviendra (B C). En fait, je pense à un peintre qui aurait à 'travailler des masses égales' pour les intégrer au reste de son dessin. Si aucun objet n'a le même état ou que toutes les propositions appartiennent déjà au pattern, on prendra alors des objets de la dernière proposition du contexte : le peintre continue à développer son dessin.
- Toute cette première partie n'a pas été programmée par manque de place en mémoire, mais elle n'offre aucune difficulté -

2 - <u>Quelle relation appliquer à ces deux objets.</u>

Cette relation sera déterminée par le pattern du contexte : Un peintre construit sa toile en appliquant certaines touches qui doivent s'harmoniser avec le reste de son dessin. Le programme en fait autant: En effet il devra trouver la relation x entre deux objets (C et B dans notre exemple) telle que :

- A partir de Cx on retrouve ce qui est lié à C dans le pattern.
- A partir de B$\bar{x}$ ($\bar{x}$ inverse de x) on retrouve ce qui est lié à B dans le pattern.

C'est à dire que si un objet α est lié à l'élément du pattern (α γ) alors, à la proposition α x devra correspondre un (/ x α) dans le contexte.

Ex : Cx ⟶ rien (et strictement rien)
B$\bar{x}$ ⟶ (B C)

Ce qui est lié aux deux objets OX et OY dans le pattern sera fourni par la fonction TRAJET .

```
(DE TRAJET  ( OX OY ;; OZ OW )
          (FOR-EACH ( !OZ !OW )
            (COND
                 ((EQ OZ OX)
                  (SETQ R-OX
                       (CONS(LIST OZ OW) R-OX)))
                 ((EQ OZ OY)
                  (SETQ R-OY
                       (CONS(LIST OZ OW) R-OY)))
            )))
```

.../...

Puis on essayera toutes les relations possibles jusqu'à ce qu'il y en ait une qui convienne.

```
(DE ESSAI    ( OX OY ;; P PREDIC )
             (SETQ AJOUT (SETQ R-OX (SETQ R-OY NIL)))
             (TRAJET OX OY)
             (PRINT =( TRAJET ,R-OX ,R-OY ))
             (SETQ PREDIC SIT)
             (WHILE PREDIC
                    (SETQ P (NEXTL PREDIC))
                    (AND (OR (NONSIT OX P R-OX)
                             (IS OX P R-OX))
                         AJOUT
                         (ASSUMING @AJOUT
                                   (OR (NONSIT OY (NON P) R-OY)
                                       (IS OY (NON P) R-OY)))
                         (NULL AJOUT)
                         (PRINT (ADD '( DO @OX @P @OY )))
                         ))
              (PRINT '//////))
```

Pour chaque essai le programme supposera que la proposition (C x B) est la bonne et pour cette raison il créera un contexte hypothétique (ASSUMING ......) composé du contexte courant <u>plus</u> l'état de l'objet défini par supposition c'est à dire (/ $\bar{x}$ C) : le peintre ajoute un élément à son dessin <u>puis</u> regarde s'il n'en détruit pas l'harmonie.

```
(DE REPONS  ( OX P )
   (SETQ AJOUT
         (COND
             (AJOUT NIL)
             ((LIST '/ (NON P) OX)) ) ))
```

ramène la liste (/ $\bar{x}$ OX).

.../...

```
(DE NONSIT  ( OX P ROX ;; OZ )
            (OR ROX
                (PRESENT '( / @P !OZ ))
                (REPONS OX P)))
```

Si l'objet OX n'est lié à aucun élément du pattern
c'est à dire Cx ⟶ rien .

```
(DE IS      ( OX R ROX ;; OZ HYP )
            (COND
                (ROX (FOR-EACH ( / @R !OZ )
                               (SETQ HYP (CONS =( @OX ,OZ ) HYP)))
                     (AND (RESPEC ROX HYP)
                          (REPONS OX R)))))
```

Essai d'une relation.

```
(DE RESPEC  ( RO HYP )
    (COND
        ((NULL RO) 'OK)
        ((MEMBER (NEXTL RO) HYP)
         (RESPEC RO HYP))))
```

Vérifie si la relation proposée respecte le pattern.
Dans notre exemple on trouvera :
(C sur B) en effet

- contexte hypothétique = contexte courant + (/ sous C)
- C sur ⟶ rien
- B sous ⟶ (B C)

.../...

puis paralellement (plusieurs relations peuvent satisfaire la même proposition)

(C à gauche de B)

Le contexte courant sera alors modifié en ajoutant définitevement :

- les propositions (C sur B)
  (C à gauche de B)
- l'état des objets mis en relation s'il ne s'y trouve pas déjà :
  (/ sous C)
  (/ à droite de C)
- le nouvel élément du pattern :
  (C B)

Tout ceci sera le travail de la fonction REMPLACE .

```
(DE REMPLACE  ( ;; OX R OY )
   (COND
       ((PRESENT '( DO !OX !X !OY ))
        (FOR-EACH ( DO !OX !R !OY )
                  (REMOVE '( TRY ,OX VERS ,OY ))
                  (REMOVE '( DO ,OX ,R ,OY ))
                  (AND (ABSENT '( / @(NON R) ,OX ))
                       (ADD '( / @(NON R) ,OX )))
                  (AND (ABSENT '( ,OX ,OY ))
                       (ADD '( ,OX ,OY ))))
       'OK )))
```

La liste de l'état des objets sera également modifiée en y ajoutant de nouveaux états :

Ex: (C est sur)
(C est à gauche)
(B est sous)
(B est à gauche)

A partir de ce nouveau contexte on pourra développer de nouvelles propositions.

Dans le cas où aucune relation n'aurait été trouvée le programme précisera les éléments du pattern liés aux objets de la proposition, par exemple :

Si aucune relation n'est trouvée pour la proposition ($\alpha \times \beta$) celle ci sera alors insérée au contexte courant sous la forme (TRY α vers β) puis sachant que α est lié à (α γ) et que β est lié à (β γ_1) et (β γ_2) on développera chaque élément de la liste :

EXPLOR = ((α γ) (β γ_1) (β γ_2))

.../...

```
(DE SEARCH  ( EX )
   (PROG ()
      B  (OR EXPLOR (RETURN))
         (SETQ EX (NEXTL EXPLOR))
         (COND
             ((ABSENT '( TRY @(CAR EX) VERS @(CADR EX) ))
              (PRINT (LIST 'EXPLORE (CAR EX) (CADR EX)))
              (ESSAI (CAR EX) (CADR EX)))
             ((GO B)))
         (AND (REMPLACE)
              (END)
              (RETURN 'OK))
         (GO B)))
```

Ainsi $(\alpha\ x\ \gamma)$ devient une nouvelle proposition et le programme devra trouver la relation x telle que

$\alpha\, x \longrightarrow$ (élément du pattern)

$\gamma\, \bar{x} \longrightarrow$ (élément du pattern)

Si une relation est trouvée, alors le contexte sera modifié, puis on essayera à nouveau la proposition $(\alpha\ x\ \beta)$

```
(DE END      ( OX OY )
   (PROG ()
      A  (COND
             ((ABSENT '( TRY !OX VERS !OY ))
              (RETURN 'OK))
             (T (FOR-EACH ( TRY !OX VERS !OY )
                          (PRINT CURRENT)
                          (ESSAI OX OY))
                (OR (REMPLACE) (RETURN))
                (PRINT '------)
                (GO A)))))
```

Cette fonction ira chercher toutes les propositions du type (TRY α vers β) et essayera de les développer.

.../...

En conclusion, le programme ne s'arrêtera que lorsque tous les objets auront les mêmes états ou lorsqu'il n'y aura plus aucune possibilité de développement. On obtiendra alors une <u>partition</u> .

D'autre part, en raison de l'absence de la fonction (décrite en 1 - du b) page 2) permettant le lancement du programme, on devra préciser pour cela le ou les couples d'objets à développer en les insérant au contexte initial sous la forme (TRY α vers β).

```
(DE DEVELOPPE  ( OX OY ;; LPOS )
   (COND
       ((END) 'NEW)
       (T (SETQ LPOS (FETCH '( TRY !OX VERS !OY )))
          (ESCAPE EXIT
                 (WHILE LPOS
                        (TRY-NEXT LPOS)
                        (PRINT =( EXPLORER ,OX VERS ,OY ))
                        (SETQ R-OX (SETQ R-OY NIL))
                        (TRAJET OX OY)
                        (SETQ EXPLOR (NCONC R-OX R-OY))
                        (AND (SEARCH (EXIT 'NEW))))))))
```

.../...

c) La partition

Cette partition sera composée d'un certain nombre de propositions réparties suivant un ordre bien précis correspondant à l'ordre des possibilités de développement. A partir de l'idée de base donnée en exemple on obtiendra la partition suivante :

(A GAUCHE B)
(A GAUCHE C) idée de base
(B SUR C)

(C SUR B)
(C GAUCHE B)
(C SOUS B)
(A SOUS B)
(B SUR A)
(B SOUS A)

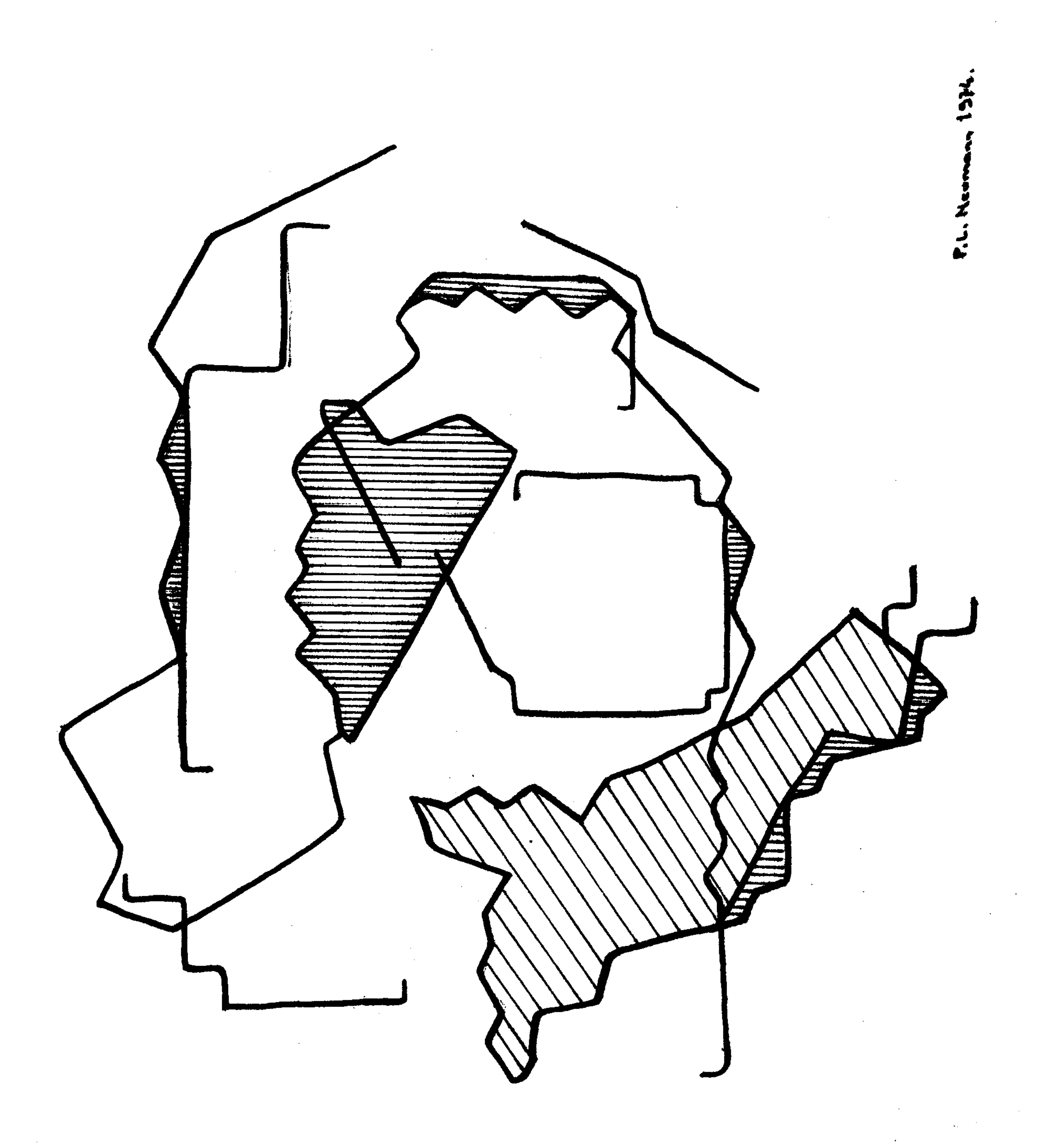
(B SOUS A)
(A GAUCHE C) idée de base
(C SUR B)

(B DROITE C)
(C SUR A)
(C GAUCHE A)
(A SOUS C)
(A DROITE C)

(C GAUCHE A)
(B DROITE C) idée de base
(A SOUS B)

(A GAUCHE B)
(B DROITE A)
(A GAUCHE C)
(C DROITE A)
(B DROITE A)

qu'il faudra INTERPRETER ....


P.L. Neumann 1974.

P.L. Neumann 1974